修 士 論 文 の 和 文 要 旨

研究科・専攻	大学院情報システム学研究科 情報ネットワークシステム学専攻　博士前期課程		
氏　　名	TSEND MUNKHDELGER	学籍番号	1252023
論 文 題 目	English alphabet based Mongolian IME		

要　　旨

モンゴル語ではキリル文字を用いている。キリル文字を直接入力するためにモンゴル言語のキリルキーボードは存在するが、配列を覚えにくく、直感的ではなく、文字は見つけにくいという問題がある。だから、キリルで直接入力できないまたは慣れないためほとんどの人はアルファベットで音訳して入力するのは多い。その結果としてモンゴル文章はキリルよりアルファベットで作成されたり、省略した・間違った単語が流通されたりする。これはサーチ・翻訳などの言語処理の発展、普及に妨げになるだけではなくキリル文字を使うモンゴルの文化遺産を壊す危険を及ぼしている。

そこで、本研究ではアルファベット文字列入力に対して正しいモンゴル語単語を作成する音訳器を提案した。既存の音訳器はウェブサイト上に使われるものが主流で、音訳の結果は 1 つに決まる決定論的なものだったのに対して本研究では日本語のカナ漢字変換に用いる統計的なモデルをキリル音訳に適応して確率の高い N ベスト候補を抽出した。また、入力が終わってからスペルチェックソフトウェア（有料版）を使って訂正する面倒な作業の代わりに入力中に直接スペルチェックできるようにした。

提案手法の最終形態として下記の機能を持つモンゴル語入力方法を作った。

① 統計的なモデルを用いて、アルファベット入力に対して最もモンゴル言語らしいキリル音訳候補 N 個作成
② より正しい単語、文章の普及のために、モンゴル言語単語辞書などを用意し、キリル音訳候補に対してスペルチェックを行う
③ よく間違う・省略する単語に対する正しい単語を登録し、入力中に訂正する
④ 予測することでユーザーの入力負担を軽減する
⑤ これらの単語候補を利用頻度、スペルチェックの回数などの要素でランキングして表示する

提案手法の検証により、提案手法の音訳機能、予測機能は一定時間をかかるのに対して、スペルチェックは入力文字列に長さに比例した時間がかかることがわかった。さらに、最大長さ40でも0.12秒で十分高速であることがわかった。また、Windows OS上の様々なアプリケーションと連動して動くことが確認できて実用性が高いものになったと考えられる。

平成２５年度修士論文

【English alphabet based Mongolian IME】

電気通信大学大学院情報システム学研究科

情報ネットワークシステム学専攻

学 籍 番 号：1252023

氏　　名：TSEND　MUNKHDELGER

主任指導教員：笠井裕之　准教授

指 導 教 員：森田啓義　教授

指 導 教 員：小川朋宏　准教授

提 出 年 月 日：平成２６年２月２０日（木）

内容概要

本論文ではアルファベット文字列入力に対してモンゴル語単語らしいキリル文字列をモンゴル語入力方法を提案する。

既存の音訳器はウェブサイト上に使われるものが主流で、音訳の結果は 1 つに決まる決定論的なものだったのに対して本研究では日本語のカナ漢字変換に用いる統計的なモデルをキリル音訳に適応する。統計的なモデルではキリル文字音訳変換をグラフ最短経路問題にして解き、確率の高い N ベスト候補を抽出する。また、それぞれの候補に対して予測やスペルチェックなどを行い、正しい単語を作成する。さらに、辞書を用いてモンゴル語単語らしさでランキングし、ユーザーにすすめる。

結果として、Windows OS 上で誰でも手軽に使えるソフトウェアを作った。

目次

第1章

研究背景

現在、モンゴル言語は表記としてキリル文字を用いている。パソコンでキリル文字を直接入力する方法として存在するのはキリルキーボードである。しかし、このキリルキーボードはロシア言語のキーボードをベースにしたものでモンゴル人にとって直感的ではなく、配列は覚えにくく、文字は見つけにくいという問題がある。また、キリル文字をボタンの上に印刷したパソコンはなかなか手に入らない。

そのため、ほとんどの人はキリル文字の代わりに英語のアルファベットを用いて音訳というやり方で入力している。結果としてインターネット上のモンゴル文章はキリルよりアルファベットで作成されたり、異なる音訳により様々書き方ができたり、省略したまたは間違った単語が流通されたりしている。アルファベットで音訳したモンゴル言語の情報は言語処理によって実現される検索、翻訳などの様々な機能の開発、発展、普及のさまたげになっている。

また、モンゴル言語の表記としてキリル文字を用いるモンゴル文化遺産を守るためにこの問題を解決しなければならない。

本研究では、言語処理でよく使う統計的なモデルを用いて、音訳して入力した英語のアルファベットの文字列をを正しいキリル文字列に変換する、モンゴル語入力方法を提案する。

第2章

関連研究

2.1 モンゴル語入力

ここで現在、主に使われているモンゴル語キリル文字を入力する方法、スペルチェックソフトウェアについて説明する [1]。

2.1.1 キリルキーボード

現在、モンゴル語キリルを入力する方法として確立されているのは Windows OS のキリルキーボードである (図 2.1)。しかし、このキリルキーボードはロシア言語のキーボードをベースにしたものでモンゴル人にとって直感的ではなく、配列は覚えにくく、文字は見つけにくいという問題がある。

図 2.1: Windows OS キリルキーボード

その他のマイナーな方法としてあるのキーの配列を新しく定義し直したキリルキーボードドライバがある。その一つ Dusal bicheech(図 2.2) はキリル文字の読みは一文字の英語のアルファベットの場合は同じ場所に配置する。マッピングできないキリル文字を余りのアルファベットの場所および記号の場所に配置している。また、CTRL+ALT の組み合わせにも配置している。キリル一文字を一つのキーにマッピングするというアイディアは良かったが、一般ユーザにとって配列を覚えるという問題を解消できていない。

Buuz キリルキーボードドライバ (図 2.3) はアルファベットは Windows OS 上のプログラムと連動する IME としてはシンプルのもので、スペルチェック・予測などはできない。アルファベットの入力をキリルに変換するには独自に変換テーブルを採用しており、いくつかのキリル文字を

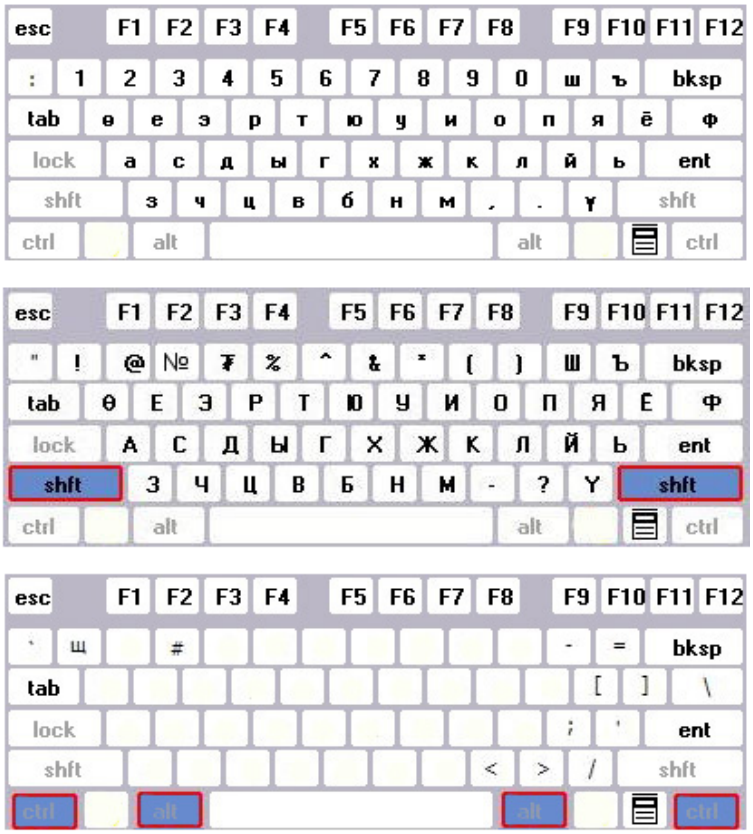

図 2.2: Dusal bicheech キリルキーボード

不自然な入力で、いくつかのキリル文字の大文字・小文字入力を連続したアルファベットで実現している。

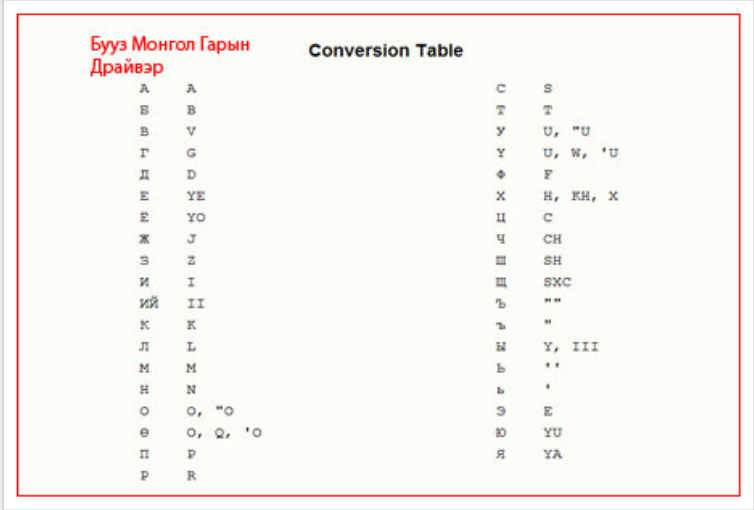

図 2.3: Buuz の変換表

入力に対して一意に決まる変換でその結果は正しいモンゴル語の単語かどうか考慮していない。

2.1.2　Web をベースにしたキリル変換器

SNS サイト、ブログなどモンゴルの Web サイト上で使えるキリル変換器もある (図 2.4, 2.5, 2.6)。モンゴル語の情報を提供する側としてもキリル変換器は必需品となっていることがわかる。同時に分かることはそれぞれ独自の変換ルールを持っていることである。これらの問題点は Web 上に特定のサイト上でしか使いないこと、Web browser 以外のアプリケーションに文章そのものをコピーペーストしなければならないことである。

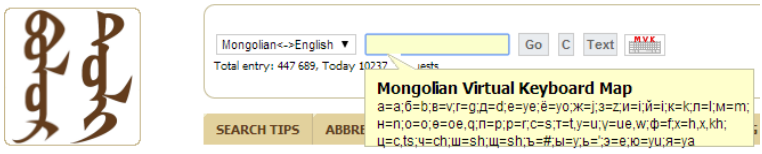

図 2.4: Bolor 英語・モンゴル語辞書サービス上の変換器

これまで紹介したすべての変換器に共通している問題は重複する音訳が存在するのに入力に対して出力が一意に決まることと訂正機能はないことである。

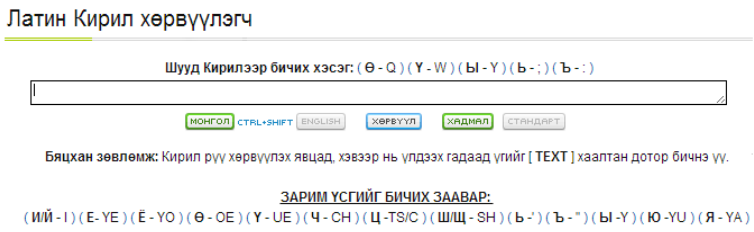

図 2.5: Medeelel ニュース・情報サイトの変換器

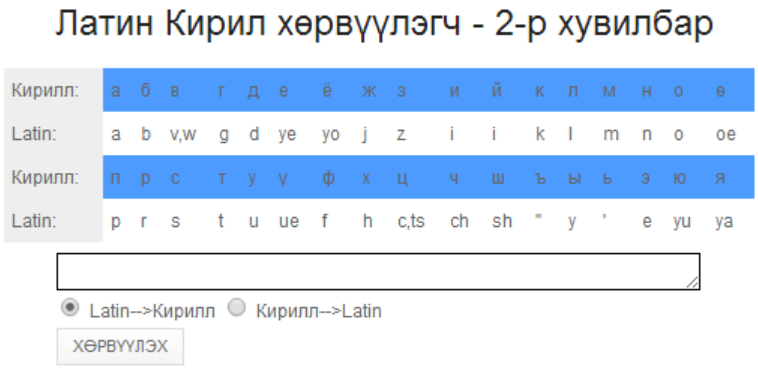

図 2.6: Tsahim urtuu ニュース・情報サイトの変換器

2.1.3　スペルチェックソフトウェア

正しいモンゴル語の単語を入力するために必要となるのはスペルチェックソフトウェアである。現在、モンゴル語スペルチェックソフトウェアとしてあるものは全て Microsoft Office 製品、OpenOffice 製品などの Plug-in として動くものである [2]。有料で一番最初に出たのはＡｎｇｕｙｕｃスペルチェックソフトウェア (図 2.7)、最新の有料版の訂正するソフトウェアはＢｏｌｏｐがある (図 2.8)。無料は OpenOffice のモンゴル語拡張辞書がある (図 2.9)。

これらのソフトウェアはワードのスペルチェック機能に似ており、入力が終わってから使うものである。モンゴル語の正しい単語を入力するため入力ソフトウェアがスペルチェック機能を持つことが一番いい解決方法である。

2.2　IME

IME とは入力方法編集プログラムのことである [3]。キーボードから直接入力することのできない文字、手書き、音声などで入力を可能にするソフトウェアである。また、複数ストロークのキー操作で 1 文字を入力するなどの仕組みを持つ。身近な例として日本語 IME がある。日本語 IME は読みとしてカナをなんらかの形で入力しておいて、漢字・かな・英字などの変換候補から選択して入力できるものである。

図 2.7: Ａｎｇｕｕｃｈ 最初のスペルチェックソフトウェア

図 2.8: Ｂｏｌｏｒ 最新のスペルチェックソフトウェア

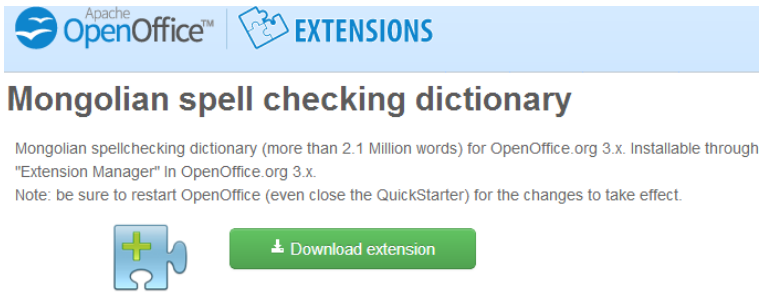

図 2.9: OpenOffice 拡張辞書

IME は主に以下の機能を持つ。

- 音訳器（transliteration）：アルファベットの読みから変換する
- 手書き (図 2.10)：ソフトウェアパッド上にマウスなどで、タッチスクリーンやタッチパネル上に指先で描いたものから変換する
- 音声認識 (図 2.11): マイクロホンなどからの音声を文章に変換する
- 予測 (図 2.12)：変換したデータからユーザの入力しようとしている文字列を予測して提案する
- 訂正：ユーザの入力を解析し、誤りを訂正する

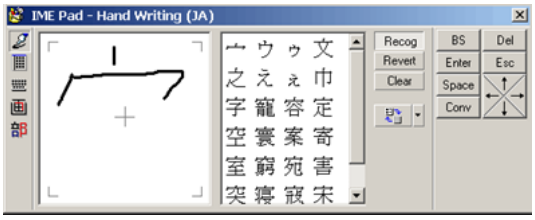

図 2.10: Microsoft 日本語入力　手書きで文字を入力する様子

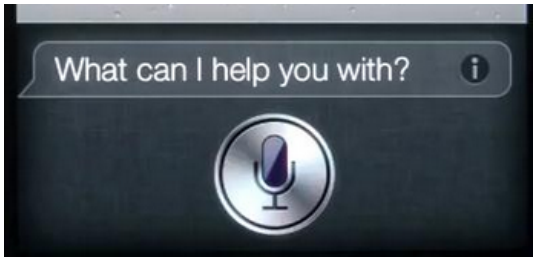

図 2.11: Iphone　Siri　音声認識ソフト

図 2.12: Google 日本語入力　かな漢字変換後さらに絵文字変換を行う様子

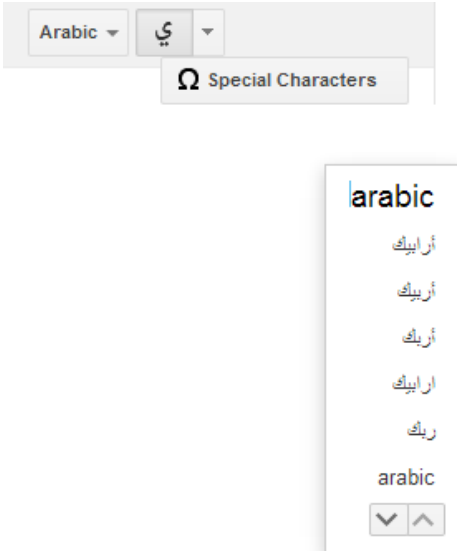

図 2.13: Google　音訳器　アラビア語文字を入力する様子

現時点で、Windows OS 用に Microsoft 社は日本語、韓国語、中国語などの高機能の IME を提供しているがモンゴル語 IME、音訳器はない。一方、Google 社はアジアだけではなくフランス語、イタリア語、スペイン語、ドイツ語向けに IME を提供している。また、ロシア語、アラビア語 (図 2.13)、ヒンディー語など数多くの言語向けの音訳器がある。

しかし、モンゴル語はキリルキーボード入力（バーチャルキーボードと呼んでいる）だけになっている。

今まで、キーボードのキーより数が多い文字（漢字など）を扱うアジア多国向けに IME が開発されていたが、最近になって欧米の多国言語も IME, 音訳器を要するようになっている。

これらの言語の IME、音訳器と同じレベルのモンゴル語 IME は必要となっている。

第3章

実装方法

実装方法の概要

提案する方法は以下の３つの手順からなる。

1. **音訳器:** 入力されたアルファベットの読みを統計的なモデル用いてモンゴル語キリル字に変換する
2. **音訳訂正:** 音訳の誤り訂正として N ベスト解を求める。さらに、よく間違う、省略する単語を列挙し、正しい単語に追加する
3. **スペルチェック・予測:** モンゴル言語の単語辞書などを用意し、スペルチェック、予測できるようにしてユーザーの入力負担を軽減する

提案手法の流れを図 3.1 に示した。

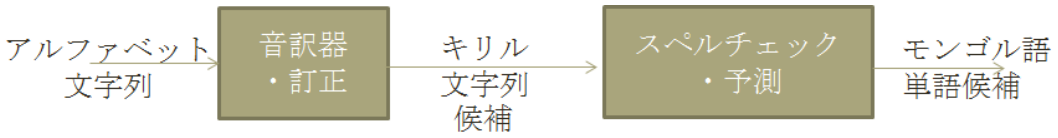

図 3.1: 提案手法の流れ

以下に提案手法の新規性と関連研究との相違点を示す。

- 日本語のひらがなからカナ漢字変換する統計的なモデル [4] をアルファベットからキリル文字に音訳するようにモデル化し、実装システムに統合した。
- 英語単語の訂正するスペルチェック [5] をモンゴル語単語に適応し、さらに編集距離と単語頻度を合わせてランキングするようにして、実装システムに統合した。
- 既存のキリル音訳器は一対一の音訳変換表を用いるのに対して複数対複数の変換でも対応できるようにした。
- よく間違う・省略する単語を列挙し、正しい単語を普及できるようにした。
- モンゴル言 IME としてはじめて、入力時にスペルチェック、予測できるようにした。

3.1 音訳器

モンゴル語キリル字とアルファベットの読みは一対一ではない (表 3.1)。例えば、キリル文字’и’と’й’の読みは”i”、’с’の読みは”c”か”ѕ”、’х’の読みは”h”か”kh”である。入力されたアルファベットは”ts”の場合はあり得るキリル文字列は”ц”,”т с”に、”khi”の場合は”х и”,”х й”,”к х и”,”к х й”になる。入力アルファベットが長ければ長いほど対するキリル文字列の数は多くなることが予想できる。しかし、例をよく見て見ると”х й”、”к х й”とう部分文字列はモンゴル語単語にない。なぜなら’й’というキリル文字はモンゴル語文法的に’а’,’э’,’о’,’у’,’и’,’’といった文字の後にしか入らない。このようなありえないケースを除き、モンゴル語らしいキリル文字列を生成する音訳器を作る方法を考える。

表 3.1: キリル音訳表

キリル文字	読み	確率	キリル文字	読み	確率	キリル文字	読み	確率
а	a	1.0	л	l	1.0	х	h,kh	1.0
б	b	1.0	м	m	1.0	ц	ts	1.0
в	v	1.0	н	n	1.0	ч	ch	1.0
г	g	1.0	о	o,oe	1.0	ш	sh	1.0
д	d	1.0		oe,o,q	1.0	щ	shch	1.0
е	ye	1.0	п	p	1.0	ъ	xx	1.0
ё	yo	1.0	р	r	1.0	ы	y	1.0
ж	j	1.0	с	s,c	1.0	ь	x	1.0
з	z	1.0	т	t	1.0	э	e	1.0
и	i	1.0	у	u,ue	1.0	ю	yu	1.0
й	i	1.0		ue,u,w	1.0	я	ya	1.0
к	k	1.0	ф	f	1.0			

以下の内容は、”日本語入力を支える技術　変わり続けるコンピュータと言葉の世界”という本 [4] に記載の手法をキリル音訳に適用したものである。日本語入力ではひらがなからカナ漢字変換に対して統計的なモデルを用いている。この研究ではモンゴル語入力方法を実現するためには、アルファベットからキリル文字に音訳するようにモデル化し、実装システムに統合した。

3.1.1 モデル化

アルファベットからモンゴル語らしいキリルに変換する音訳器を作るために次の統計的なモデルを用いる。ユーザから入力されたアルファベット文字列を $\boldsymbol{x}$、それに対するキリル音訳候補を $\boldsymbol{y}$ としたときに $P(\boldsymbol{y}|\boldsymbol{x})$ の確率が高い $\boldsymbol{y}$ を求める。ベイズの定理より、$P(\boldsymbol{y}|\boldsymbol{x})$ は、

$$P(\boldsymbol{y}|\boldsymbol{x}) = P(\boldsymbol{x}|\boldsymbol{y})P(\boldsymbol{y})/P(\boldsymbol{x}) \tag{3.1}$$

と書き換えらる。ここで、この式の右辺の $P(\boldsymbol{y})$ をキリルモデル、$P(\boldsymbol{x}|\boldsymbol{y})$ を変換モデルとなる。$P(\boldsymbol{y})$ は変換候補 $\boldsymbol{y}$ のキリル化としての正しさを表す。$P(\boldsymbol{x})$ は入力アルファベット文字列 $\boldsymbol{x}$ が決まれば一定の値になるので、次のように

$$P(\boldsymbol{y}|\boldsymbol{x}) \propto P(\boldsymbol{x}|\boldsymbol{y})P(\boldsymbol{y}) \tag{3.2}$$

を最大とする $\boldsymbol{y}$ を求めても同じ結果が得られる。この式からわかることはキリルモデル、変換モデルの両方の値が高い $\boldsymbol{y}$ が変換結果として望ましいということである。

3.1.2　キリルモデル

キリルモデル $P(\boldsymbol{y})$ は、文字列 $\boldsymbol{y}$ がモンゴル語らしければ高い値を、モンゴル語らしくなければ低い値を割り当てるモデルです。ここで、キリル文字列 $\boldsymbol{y}$ をキリル文字 y_0、 y_1、...、y_n からできているとして次のように分解できると仮定する。

$$P(\boldsymbol{y}) = P(y_0) \times P(y_1|y_0) \times \cdots \times P(y_i) \times P(y_{i+1}|y_i) \times \cdots \times P(y_n|y_{n-1}) \times P(y_n) \tag{3.3}$$

ここで、$P(y_{i+1}|y_i)$ はキリル文字 y_i の後にキリル y_{i+1} が入る接続確率である。また、$P(y_i)$ はキリル文字の出現確率であるが本研究では $P(y_i) = 1$ と見なす。従って、キリルモデルは以下の式で表される。

$$P(\boldsymbol{y}) = P(y_1|y_0) \times \cdots P(y_{i+1}|y_i) \times \cdots \times P(y_n|y_{n-1}) \tag{3.4}$$

3.1.3　接続確率の推定

キリル文字 a の後にキリル文字 b が入る接続確率 $P(b|a)$ を次のように求める。モンゴル語単語辞書を用意し、すべての単語の隣接するキリル文字 a と b に対して頻度 $Count(b|a)$ を数える。キリル文字 a から始まる二文字の頻度 $Count(a)$ は以下のように求める

$$Count(a) = \sum_{b \subset cyrillicLetters} Count(b|a) \tag{3.5}$$

すると、接続確率は以下の式て求められる。

$$P(b|a) = Count(b|a)/Count(a) \tag{3.6}$$

辞書にはないキリル文字の並び確率がゼロになるゼロ頻度問題が起きる可能性がある。そこで、以下に示すスムージング方法で少しだけ確率 $smoothingProbability$ を割り当てておく。

$$CountAll = \sum_{a \subset cyrillicLetters} Count(a) \tag{3.7}$$

$$smoothingProbability = 1/CountAll \tag{3.8}$$

キリル以外のスペース、ハイフン、括弧などの記号の場合は確率は 1 とする。また、単語の先頭にキリル文字入る確率及び単語の末尾（まつび）にキリル文字が入る確率を常に 1 とする。

3.1.4 変換モデル

キリル文字 y_i のアルファベット読みを $\boldsymbol{x}_i$ であるとする。ここで注目したいのはキリル文字一つに対してアルファベットの読みとなる一文字ではなく文字列であることです。また、$\boldsymbol{y} = y_0 + y_1 \cdots + y_n$、 $\boldsymbol{x} = \boldsymbol{x}_0 + \boldsymbol{x}_1 \cdots + \boldsymbol{x}_n$ を満たす。ここで + 記号は文字列の結合（連結）を表す。キリルモデルと同様に $\boldsymbol{y}$ はキリル文字 y_0、 y_1、...、y_n からできていると仮定すると、

$$P(\boldsymbol{x}|\boldsymbol{y}) = P(\boldsymbol{x}_0|y_0) \times \cdots \times P(\boldsymbol{x}_n|y_n) \tag{3.9}$$

になる。2 章に紹介したように既存のモンゴル語キリル変換方法はキリル文字 1 つに対して異なるアルファベットの読みを持つようになっていた。どの読みが一番よく使われているのかを推定するのは難しく、それほど重要ではないので、本研究ではすべての i に対して $P(\boldsymbol{x}_i|y_i) = 1$ とする。従って、

$$P(\boldsymbol{x}|\boldsymbol{y}) = 1 \tag{3.10}$$

になる。

3.1.5 グラフの構築 [4]

アルファベットの文字列 $\boldsymbol{x}$ の分割方法によっていくつかのキリル文字列 $\boldsymbol{y}$ があること、キリル文字列 $\boldsymbol{y}$ に対していくつかの異なる読みアルファベットの文字列 $\boldsymbol{x}$ があり得る。本研究の音訳器ではあるアルファベットの文字列 $\boldsymbol{x}$ に対して $P(\boldsymbol{y}|\boldsymbol{x})$ が最大になる音訳方法すなわち、最適なキリル文字列 $\boldsymbol{y}$ を求める。式 3.4、3.10 を式 3.2 に代入すると、

$$P(\boldsymbol{y}|\boldsymbol{x}) \propto P(y_1|y_0) \times \cdots P(y_{i+1}|y_i) \times \cdots \times P(y_n|y_{n-1}) \tag{3.11}$$

になり、この式を解けばよいことがわかる。さらに両辺の対数をとり、マイナス 1 でかけると、

$$-\log P(\boldsymbol{y}|\boldsymbol{x}) \propto -\log P(y_1|y_0) - \cdots - \log P(y_{i+1}|y_i) - \cdots - \log P(y_n|y_{n-1}) \tag{3.12}$$

この式を用いてキリル文字をノード、キリル文字同士の接続（繋がりやすさ）確率の対数を辺とするグラフを構築する。

ノードと辺の作成

キーはキリル文字、値は読みになるハッシュテーブル $conversionHash$ を用意する。同じ読みを持つ異なるキリル文字もあり得るのでハッシュテーブルは複数の値を持つものとする。

アルファベットの文字列 $\boldsymbol{x}$ に対して、なりえるキリル文字ノードを以下のアルゴリズム 3.2 で作成する。ここで、$CNode$ はノード構造体でアルファベットの文字列 $\boldsymbol{x}$ 中での始点 ($start$)、終点 $finish$ の場所、読みのアルファベット部分文字 $reading$、対応するキリル文字 $display$ で出来る。辺を以下アルゴリズム 3.3 で作成する。

```
1. def CreateNode(x, graph)
2. for(i = 0; i < x.length(); i++)
3.   for(j = max(0, i − 4); j <= i; j++)
4.     reading = x.substr(j, i − j + 1);
5.     if(conversionHash.has(reading)
6.       foreach(cyrillicletter : conversionHash[reading])
7.         CNode node;
8.         node.start = j;
9.         node.finish = i;
10.        node.reading = reading;
11.        node.display = cyrillicletter;
12.        node_list.add(node));
13. graph.add(node_list)
```

図 3.2: ノードの作成方法

```
1. def CreateEdge(x, graph)
2. for(i = 0; i < graph.nodes.size(); i++)
3.   for(j = i + 1; j <= graph.nodes.size(); j++)
4.     if(graph.nodes[i].finish + 1 == graph.nodes[j].finish){
5.       CEdge edge;
6.       edge.left = graph.nodes[i];
7.       edge.right = graph.nodes[j];
8.       edge.weight = −log(p(graph.nodes[j].display|graph.nodes[i].display));
9.       edge_list.add(edge);
10. graph.add(edge_list)
```

図 3.3: 辺の作成方法

$CEdge$ は辺構造体で始点ノード ($left$)、終点ノード $right$ の場所、重み $weight$ で出来る。重み $weight$ は式 3.12 の右辺に現れるキリルモデルの接続確率である。例として、アルファベットの文字列 $\boldsymbol{x}$ = ”$tsonkh$” の場合のグラフを図 3.4 に示す。このようにキリル文字をノード、隣接するキリル

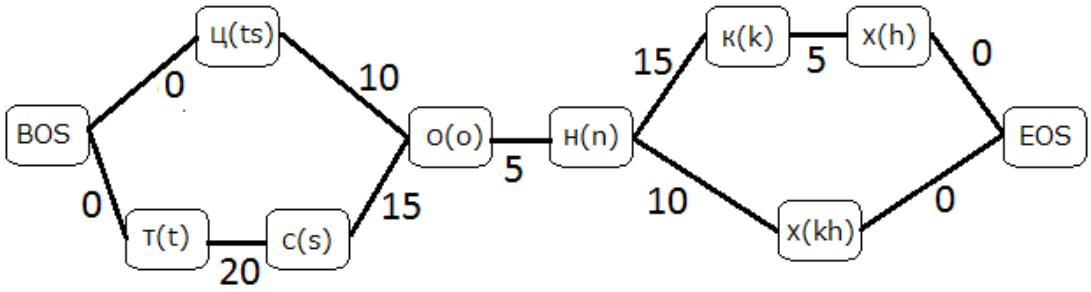

図 3.4: 入力文字列”tsonkh”に対して構築されるグラフ

文字同士に辺を辺の重みに接続確率を取るグラフを構築できる。ここで、BOS と EOS はそれぞれ単語の始まり、終わりを表すことにする。図 3.4 の例では BOS から EOS までのパスの中で一番短いのは BOS →ц (ts) →о (o) →н (n) →х (kh) →EOS のパスでコストは 0 + 10 + 5 + 10 + 0 = 25 になっている。この例ではц о н х (25), ц о н к х (35), т с о н х (40), т с о н к х (60) の 4 つのパスがあるが、モンゴル単語として意味があるのはц о н х（窓:window）だけである。このようにグラフを構築すると式 3.12 は単語の始まりから終わりまでの最短経路を求める問題になっていることが分かる。

3.1.6　最短経路問題 [4]

構築したグラフの最短経路問題を以下に示すビタビアルゴリズムと呼ばれる高速なアルゴリズムを使って解く [4], [6]。このアルゴリズム 3.5 では $node\_list$ は i 文字目で終わるノード (文字) を返すものとします。get_edge_cost は 2 つのキリル文字の接続確立のマイナス対数を辺のコストとして返す。get_prev_nodes は隣接するノードのうち、スタート側のものを返す関数である。

ビタビアルゴリズム 3.5 では、単語の始まりを表す BOS から単語の終わりを表す EOS へ向かってそれぞれのノードに対し次の操作を行う。

1. 注目しているノードの隣接ノードのうち、BOS 側に近いものを取り出す。それぞれの隣接ノードに対し、2,3 の操作を行う
2. 隣接ノードを通った場合の BOS から注目ノードまでの合計コストを、以下の 3 つの和として求める。
   - BOS から隣接ノードまでの合計コスト
   - 隣接ノードから注目ノードまでの接続確率であるコスト

```
1. def forward_viterbi(graph)
2.     for i in 1 to n
3.     #i 文字目で終わるノードを返す
4.         nodes = node_list(graph, i)
5.         for node in nodes
6.           node_cost = get_node_cost(node)
7.           cost = Double_MAX
8.           shortest_prev = false
9.           prev_nodes = get_prev_node(node)
10.          for prev_node in prev_nodes
11.              edge_cost = get_edge_cost(prev, node)
12.              tmp_cost = prev.fcost + edge_cost + node_cost
13.              if tmp_cost < cost
14.                  cost = tmp_cost
15.                  shortest_prev = prev
16.          node_prev = shortest_prev
17.          node.fcost = cost
18.
19. node = EOS
20. result = []
21. while node! = bos
22.     result.push(node)
23.     node = node.prev
24. return result.reverse
```

図 3.5: 最短経路問題を解くビタビアルゴリズム [4]

- 注目ノード自体のコスト。本研究ではキリル文字自体の出現確率を１としているから０。

3. 求めた合計コストが現在の最低コストよりも小さければ、注目ノードの $prev$ での参照をその隣接ノードへと切り替える。

ビタビアルゴリズムでは、あるノードに注目しているとき、そのスタート側の隣接ノードでは最短経路はすでに確定している。従って、すべてのノードに対して上記の手順を行ったあとは、EOS から prev を順にたどって最短経路を求められる。その結果を配列に詰め込み、reverse 関数で順序を反転してから結果を得られる。

3.2　音訳訂正

3.1 章では入力されたアルファベット文字列の読みを統計的なモデル用いて最適なキリル文字列に変換する方法を説明した。ここで、音訳の誤り訂正として N ベスト解を求める。さらに、よく間違う、省略する単語の読みを列挙し、正しいモンゴル語単語に追加する。

3.2.1　N ベスト解を求める [4]

N ベスト解を求める方法として、前向き DP(動的計画法: Dynamic Programming) 後ろ向き A*(え―すた―: A star) アルゴリズムと呼ばれる手法を使う [4], [7]。前向き DP と呼ばれる部分は 3.1 章のビタビアルゴリズムの結果を生成するところを除いたものになる。A*アルゴリズムではコストが低い順に要素を取り出せる優先度付きキューを使ってたどるノードを選んでいく。また、N ベスト解を求める時は単語の始まり BOS からそのノードまでのコスト $node.fscore$ 以外に、そのノードから単語の終わり EOS までのコストを $node.gscore$ に保存する。以下に、A*アルゴリズムを示す。A*アルゴリズムではまず、優先度付きキューにゴールノードを挿入する。その次の行からのループでは、以下のような作業を行う。

1. 優先度付きキューから最もスコアの高いノードを取り出す
2. 取り出したノードがスタートノードであった場合、そのノードを結果に追加する
3. スタートノードではなかった場合、そのノードに隣接するスタート側のノードのリストを取り出す
   (a) 取り出した各隣接ノードに対して、隣接ノードの $gscore$ を注目しているノードの $gscore$ とノード間のスコアと隣接ノードのスコアの和にする
   (b) 隣接ノードを優先度付きキューに入れる。優先度は隣接ノードの $fscore$ と $gscore$ の和とする

A*アルゴリズムでは 3a ステップで隣接ノードを優先度付きキューに登録する際に、$fscore$ と $gscore$ の和を優先度としている。つまり、優先度は「語の始まり BOS からそのノードまでのコスト」と「そのノードから単語の終わり EOS までのコスト」の和になる。ただし、後者はベスト

```
1. def backward_a_star(graph, n)
2.     results = []
3.     pqueue = PriorityQueue.new
4.     pqueue.push(EOS, 0)
5.     while !pqueue.empty
6.         node = pqueue.pop()
7.         if node.is_BOS
8.             cost = node.gscore + node.fscore
9.             word = []
10.            while !node.is_BOS
11.                word.push(node)
12.                node = node.next
13.            results.push(key = word, value = cost)
14.        else
15.            prev_nodes = get_prev_node(node)
16.            for prev in prev_nodes
17.                edge_score = get_edge_score(prev, node)
18.                prev.gscore = node.gscore + edge_score + get_node_score(node)
19.                prev.next = node
20.                #隣接ノードをコピーして優先度付きキューに入れる
21.                newnode = copy(prev);
22.                pqueue.push(newnode, newnode.gscore + newnode.fscore)
23.        if results.size > n
24.            return results
```

図 3.6: N ベスト解を求める A*アルゴリズム [4]

スコアであるとは限らない。この優先度は「そのノードからベストのルートを通って行くと、このスコアが実現できる」という実現可能な値を示す。

A*アルゴリズムでは優先度付きキューには最初にゴールノードを入れてあるので、1 回目はゴールノードだけが取り出される。そのため、ゴールノードの隣接ノードを優先度付きキューに入れる。その次にキューから取り出されるノードは明らかにベストスコアを実現するノードになる。以降、スコア的にベストなノードを取り出してその隣接ノードを優先度付きキューに入れている。この作業を続けるとそのうちスタートノードが優先度付きキューから出て来る。これはビタビアルゴリズムで求めた結果と同じものになる。その次に出てくるのは第 2 位の候補を実現するためのノードになる。第 2 位の候補は第 1 位の候補とどこからずれてくるわけで、そのずれる可能性のあるすべての場所で隣接ノードを優先度付きキューに入れているので、ありうる第 2 位の候補はすべて優先度付きキューに登録されている。第 3 位以降は第 1 位と 2 ヵ所以上の部分でずれる可能性があり、そのような候補は第 2 位の候補が求まった時点ですべて優先度付きキューに登録されている。第 4 位以降も同様である。

3.2.2　よく間違う、省略する単語

アルファベットでキリル単語の読みを書くときよく省略したり、間違ったりすることがある。例えば、”с а й н (元気) б а й н а（です） у у（か）”の正しい読みは”sain baina uu”であるが、省略して”sn bn u”と書くことが多い。提案手法の音訳器及び音訳訂正機能を使った場合結果は”с н　б н　у”となってしまう。そこで、こういう単語を登録して候補に足す機能を追加する。3.1 の前向き DP 後ろ向き A*アルゴリズムの結果に以下のように候補を追加しておく。こ

```
1. def add_abbreviation_word(results, reading)
2.     if abbreviationHash.has(reading)
3.         results.push(abbreviationHash[reading])
```

図 3.7: よく間違う・省略する機能

のように、よく間違う・省略する単語の読みをキーに、単語の表記を値にするハッシュテーブル $abbreviationHash$ を用意し、登録しておく。

3.3　スペルチェック・予測

入力文字列 $\boldsymbol{x}$ に対してもっともらしいキリル音訳 $\boldsymbol{y}$ の N ベスト解を求める方法を 3.2 章に説明した。ユーザーの入力ミスを訂正するためにスペルチェックする機能とユーザーの入力負担を軽減するために予測する機能を追加する。

以下のスペルチェック実装内容は，”How to Write a Spelling Corrector”という記事 [5] に記載の手法をモンゴル語に適用したものである。但し、ランキングについてはモンゴル語入力方法を

実現するために、編集距離と単語頻度を合わせたものを提案し、実装システムに統合した。

3.3.1 スペルチェック [5]

キリル音訳文字列 $\boldsymbol{y}$ に対して最も可能性の高いモンゴル語スペルチェック候補を選びたいとする。そのために可能な修正候補の集合の中から確率 $P(\boldsymbol{z}|\boldsymbol{y})$ が最大になるモンゴル語単語 $\boldsymbol{z}$ を見つける。ベイズの定理により、$P(\boldsymbol{z}|\boldsymbol{y})$ は、

$$P(\boldsymbol{z}|\boldsymbol{y}) = P(\boldsymbol{y}|\boldsymbol{z})P(\boldsymbol{z})/P(\boldsymbol{y}) \tag{3.13}$$

である。$P(\boldsymbol{y})$ はどの $\boldsymbol{z}$ に対しても同じなので、ここでは無視することができる。従って、

$$P(\boldsymbol{z}|\boldsymbol{y}) \propto P(\boldsymbol{y}|\boldsymbol{z})P(\boldsymbol{z}) \tag{3.14}$$

を最大にする $\boldsymbol{y}$ を求めても同じ結果が得られる。この式からわかることは言語モデル $P(\boldsymbol{z})$、誤りモデル $P(\boldsymbol{y}|\boldsymbol{z})$ の両方の値が高い $\boldsymbol{z}$ がスペルチェック訂正の結果になる。

言語モデル

言語モデル $P(\boldsymbol{z})$ は修正語 $\boldsymbol{z}$ 自体の確率で、単語 $\boldsymbol{z}$ がモンゴル語単語としてのどれくらい入力される可能性があるかを表すものである。例えば P(” б а й н а ”) は比較的確率が高いのに対し、P(” б н б н б н у у ”) はほとんど 0 に近い (б а й н а:です、であるの意味を持つ、б н б н б н у у：無意味のキリル文字の並び)。言語モデル $P(\boldsymbol{z})$ を以下のように求めることができる。文法的に正しいモンゴル語文章を用意し、すべての単語 $\boldsymbol{z}$ の頻度 $Count(\boldsymbol{z})$ を数える。文章の全単語数を $CountWords$ とする。すると、

$$P(\boldsymbol{z}) = Count(\boldsymbol{z})/CountWords \tag{3.15}$$

問題は未知の単語の場合は確率が 0 のなるゼロ頻度問題が起きることである。そこで、未知単語 $\boldsymbol{z}$ の追加の場合その単語の頻度を $Count(\boldsymbol{z}) = 1$ として少しだけ確率割り当てることにする。すべての単語に対して $CountWords$ が一定であるので無視できる。

$$P(\boldsymbol{z}) \propto Count(\boldsymbol{z}) \tag{3.16}$$

誤りモデル

誤りモデル $P(\boldsymbol{y}|\boldsymbol{z})$ は単語 $\boldsymbol{z}$ を意図して書くが、単語 $\boldsymbol{z}$ が入力されてしまう可能性を表す確率である。ユーザーがよくする誤りとしてあげられるのは以下の通りである [5]。

- 一文字を入力し忘れる
- 一文字を多く入力する
- 違う一文字を入力する

- 2 つの文字の順番を逆に入力する

正しい単語 $\boldsymbol{z}$ からどの程度異なる単語 $P(\boldsymbol{y}$ なったかを表す数値としてあげられるのは 2 つの単語の間の編集距離（レーベンシュタイン距離）$distance(\boldsymbol{y}, \boldsymbol{z})$ である。簡単に言うと、上記の誤りでは「2 つの文字の順番を逆に入力する」場合の編集距離を 2, それ以外の場合は 1 となる。編集距離は短いほど $P(\boldsymbol{y}|\boldsymbol{z})$ は高く、長いほど低くなる。本研究では、編集距離を用いて $P(\boldsymbol{y}|\boldsymbol{z})$ を以下のように定義する。

$$P(\boldsymbol{y}|\boldsymbol{z}) \propto 1/(1 + distance(\boldsymbol{y}, \boldsymbol{z})) \tag{3.17}$$

ここで、言語モデル式 3.16 と誤りモデルの式 3.17 を 3.14 に代入する。

$$P(\boldsymbol{z}|\boldsymbol{y}) \propto Count(\boldsymbol{z})/(1 + distance(\boldsymbol{y}, \boldsymbol{z})) \tag{3.18}$$

この式から編集距離が少なくかつ出現頻度高い単語 $\boldsymbol{z}$ を見つけるのがスペルチェックになることを示している。

スペルチェック単語の抽出

考えられる誤り 3.3.1 章に紹介した通りであるとした時は、キリル音訳 $\boldsymbol{y}$ のスペルチェック単語を作成するのは容易である。以下にキリル音訳 $\boldsymbol{y}$ に対してスペルチェックを行い結果を式 3.18 でソートしてして返すアルゴリズムを示す。このアルゴリズムでは $\boldsymbol{y}$ に対して考えられる誤りと逆の操作をすることでのスペルチェックを行っている。$MongolianDictionary$ モンゴル語単語辞書であり単語をキーに、頻度を値に持っている。また、$cyrillicLetters$ はモンゴル語キリル文字の集合である。スペルチェックの操作は以下の通りである。

- 一文字を入力し忘れたという仮定で一文字を挿入する。この時、挿入できる場所は先頭から末尾まであり、挿入する文字は $cyrillicLetters$ に属する。
- 一文字を多く入力したという仮定で一文字を削除する。この時、削除できる文字は $\boldsymbol{y}$ の文字である。
- 違う一文字を入力したという仮定で一文字を変更する。この時、$\boldsymbol{y}$ の文字を $cyrillicLetters$ に属する文字に変更する。
- 2 つの文字の順番を逆に入力したという仮定で隣接する 2 つの文字を置換する。この時、$\boldsymbol{y}$ の中で隣同士の文字を置換する。

このような操作で作られた修正文字列 $\boldsymbol{z}$ をモンゴル語単語辞書 $MongolianDictionary$ にあれば、式 3.18 の値で登録する。最後にこれらの修正文字列 $\boldsymbol{z}$ を値でソートして返すことでスペルチェックを行う。

```
1. def spellcheck(y)
2.    l = len(y)
3.    s = []
4.    #変更なし
5.    if !s.has(z) and MongolianDictionary.has(z)
6.       s.insert(key = z, value = MongolianDictionary[z])
7.    #挿入
8.    foreach(c : cyrillicLetters)
9.       for(i = 0; i <= l; i++)
10.         z = y.substr(0, i) + c + y.substr(i)
11.         if !s.has(z) and MongolianDictionary.has(z)
12.            s.insert(key = z, value = MongolianDictionary[z]/2)
13.   #削除
14.   for(i = 0; i < l; i++)
15.      z = y.substr(0, i) + y.substr(i + 1)
16.      if !s.has(z) and MongolianDictionary.has(z)
17.         s.insert(key = z, value = MongolianDictionary[z]/2)
18.   #変更
19.   for(i = 0; i < l; i++)
20.      z = y
21.      foreach(c : cyrillicLetters)
22.         z[i] = c
23.         if !s.has(z) and MongolianDictionary.has(z)
24.            s.insert(key = z, value = MongolianDictionary[z]/2)
25.   #置換
26.   for(i = 0; i < l - 1; i++)
27.      z = y
28.      c = z[i]
29.      z[i] = z[i + 1]
30.      z[i + 1] = c
31.      if !s.has(z) and MongolianDictionary.has(z)
32.         s.insert(key = z, value = MongolianDictionary[z]/3)
33.   sort s by value
34.   return s
```

図 3.8: スペルチェックアルゴリズム [5]

3.3.2 予測

IME には音訳、スペルチェック以外には予測という重要な機能がある。スペルチェックでは単語の入力が完了したという仮定でしたが、ユーザーがまだ単語を最後まで入力していない場合入力しそうな単語を予測して提案することで入力負担を軽減できる。入力文字列 $\boldsymbol{x}$ に対してのキリル音訳 $\boldsymbol{y}$ ができたとする。その中で最も確率が高いキリル音訳 $\boldsymbol{y}$ に対して予測単語単語 $\boldsymbol{w}$ を求める。この時、$\boldsymbol{y}$ は $\boldsymbol{w}$ の部分文字列で先頭から始まるものである。$\boldsymbol{w}$ は単語辞書に属するので以下アルゴリズムで探せる。$lower\_bound$ はソート済みの系列に対して，指定した要素以上の値を

1. $def\ prediction(y)$
2. $\quad l = len(y)$
3. $\quad result = []$
4. $\quad current = lower\_bound(MongolianDictionary, y)$
5. $\quad w = current.key$
6. $\quad while(w.substr(0, l) == y)$
7. $\quad\quad result.insert(current)$
8. $\quad\quad current = current.next$
9. $\quad\quad w = current.key$
10. $\quad sort\ result\ by\ value$
11. $\quad return\ result$

図 3.9: 予測機能

持つ最初の要素を返する関数で、ほとんどのプログラミング言語の標準ライブラリに入っている。このアルゴリズムではまず、モンゴル語単語辞書の中から文字列 $\boldsymbol{y}$ 以上のキーを持つ要素 $current$ を探している。そして、その要素の最初の l 文字の部分文字列は $\boldsymbol{y}$ と同じければ $result$ に追加し、次の要素に進んでいる。最後の $result$ を頻度でソートして返すことで予測を行う。

3.4 ランキング

ここで、入力されたアルファベット文字列読み $\boldsymbol{x}$ に対して提案手法で作成される候補を整理する。

- 音訳器によって変換される N 個のキリル文字列 $\boldsymbol{y}$
- スペルチェックによって訂正されるモンゴル語単語 $\boldsymbol{z}$。（N 個のキリル文字列 $\boldsymbol{y}$ それぞれに対してスペルチェック行う）
- 予測機能によって提案するモンゴル語単語 $\boldsymbol{w}$。（最も確率高いキリル文字列 $\boldsymbol{y}$ に対して予測が行わる）
- よく間違う・省略する単語。

これらの候補をランキングしてユーザーにすすめたいがここで考慮したいことはモンゴル語単語辞書にない未知単語対策である。辞書にはない単語も入力できるようにするため、音訳器の結果をいつも出したい。その中で選びやすくするため最も確率高いキリル文字列を最初に出す必要がある。

以上のことから、よく間違う・省略する単語候補があれば第 1 候補に、最も確率高いキリル文字列 $\boldsymbol{y}$ を第 2 候補に、キリル文字列 $\boldsymbol{y}$ の中で辞書に存在するものを、そして予測結果 $\boldsymbol{w}$ を、その次からスペルチェック結果、最後に辞書になかったキリル文字列 $\boldsymbol{y}$ を入るように順位付ける。

3.5　モンゴル語 IME の実装

提案手法の最終形態として英語のアルファベットをベースにしたモンゴル語入力方法（English alphabet based Mongolian IME）を作った。

Microsoft の IME 実装方法のサンプルソースコード [3] を参考に作ったが、実装方法と注意した点を簡単にまとめると以下の通りである。

- 予め決められた方法で実装する必要がある。
- コードをコンパイルして、できた dll 拡張のプログラムを Windows のシステム上の登録 (regsvr32 textservice.dll) することで初めて IME として機能する。
- アプリケーション上に IME が有効になるとこの dll が読み込まれる。これによって Windows 上のどの Application と連動して動くようになる。IME はクラッシュするとアプリケーションもクラッシュするので注意しなければならない。また、IME はアプリケーションに読み込まれて起動するのでできるだけ小さいサイズのほうが望ましい。
- キー入力、マウスの動き、アプリケーションの切換えなどすべてのエベント（操作）が IME に渡され、それに対して IME は入力、テクスト編集、候補 Window などを行う

モンゴル IME を実装するために新たに追加したものは以下の通りです。

- 追加される IME はモンゴル言語から選べられる
- 入力されるキーをローマ字にして、キリル文字に音訳する
- 辞書ファイルを読み込み、単語を検索する
- スペルチェック、予測機能を追加する
- 候補 Window から候補を選べたり、複数の候補を表示する
- アルファベットの入力も候補に追加（最後の候補）
- Windows 上のほとんどの Application と連動して動くようにする。

図 3.10 に作成したモンゴル語入力方法使っている様子を示す。右側には Chrome broswer の検索フィールドで入力し、音訳器の候補だけが現れた様子である。左側には Word で入力し、予測

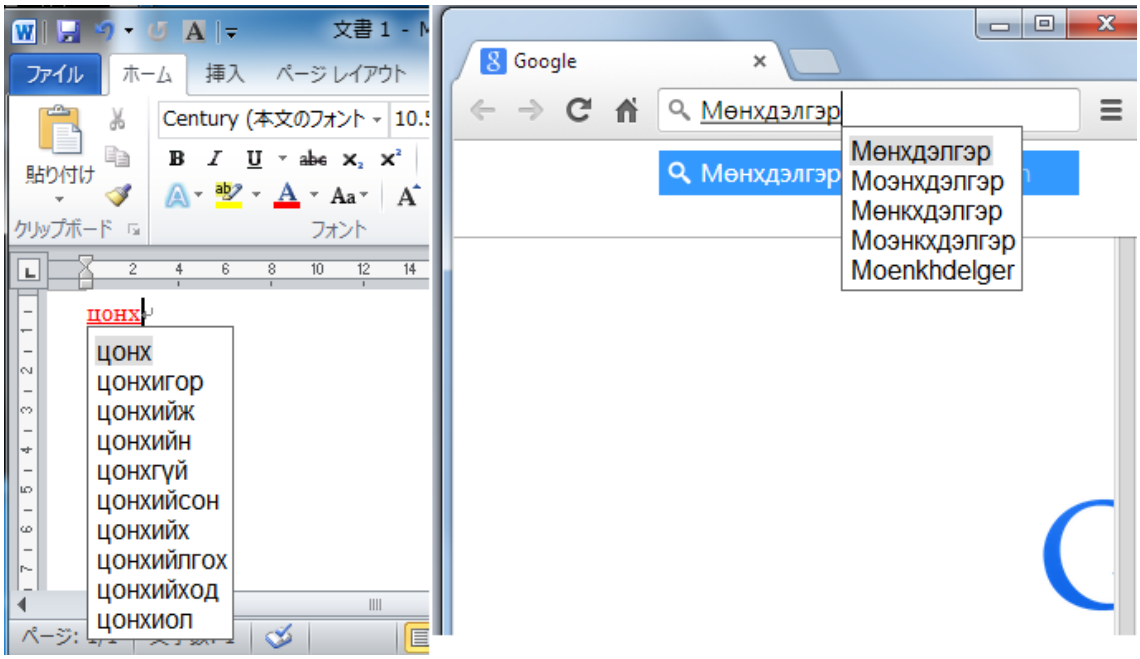

図 3.10: モンゴル語入力方法の使用例

結果が出ている様子である。この様にモンゴル語入力方法として提案手法がちゃんと動いていることがわかる。

第4章

モンゴル語入力方法の検証

4.1 計算量

提案手法は入力に対して音訳、スペルチェック、予測などを行っている。もし、これらの機能が遅すぎるとユーザーにはアプリケーションが止まっているように感じ、実用性が低いことになる。そこで、提案手法は十分な高速であることを示す。

4.1.1 音訳器

音訳器で構築されるグラフのノード数はキリル文字と読みの組み合わせで決まる。音訳表 3.1 を見ると読みな最大な長さは 4、同じ読みを持つ最も多いキリル文字は 2 つある。入力文字列の長さ L とした時に、最大ノード数は $8L$ になる。グラフの辺の数は隣接するノードの数で決まるので最大で $8 * 8 * L = 64L$ である。ビタルビアルゴリズムの計算量は辺の数で決まるので $O(64L)$ になる。優先度付きキューの処理は辺の数の対数の計算量なので $O(log(64L))$ になる。エースターアルゴリズムで N 個の候補を抽出すると計算量は $O(N * 64L * log(64L))$ になる。モンゴル語 IME では $N = 20$, 最大長さ $L = 40$ なので十分高速であると言える。

4.1.2 スペルチェック

キリル文字の数は 35 なので、挿入が $35(L + 1)$、削除が L、変更が $35L$、置換が $L - 1$ 通りであり、合計 $72L + 34$ 通りである。N 個のスペルチェックを行うとして候補数は $N * (72L + 34)$ になり、スペルチェック候補作成の計算量は $O(N * (72L + 34))$ である。モンゴル語単語辞書 $MongolianDictionary$ の単語数 M として、スペルチェック候補それぞれを辞書にあるかどうかを確認する計算量は $O(N * log(M) * (72L + 34))$ これは、ユーザーが 1 回だけ誤って入力したと仮定であり、もし 2 回ミスしたとした場合は計算量 $O(N * log(M) * (72L + 34) * (72L + 34))$ となり、遅くなる可能性があることがわかる。より良いスペルチェックを行うために想定するミスの数を増やす必要があるが、モンゴル語 IME として実用性を考えると少ないほうが良い。

4.1.3 予測

$lower\_bound$ の関数の実装は二分探索に似ているので計算量は $O(log(M))$ になる。さらに、条件を満たす候補は K とすると、計算量は $O(log(M) + K)$ になる。入力文字の数は少なかったり、

辞書の単語数は多い時は K は M に比例してしまうのでどこかで打ち切らないといけない。提案手法では $K = 10$ としている。

4.1.4 実験

実際に計算時間を計測してみる。理論上の音訳候補が多くになる入力を考える。表 3.1 からキリル文字’ т ’の読みは”t”、’ с ’の読みは”s”、’ ц ’の読みは”ts”であるので、入力文字列”ts”に対して音訳候補は” ц”, ” т с ”の 2 通りである。入力文字列は”ts”連結したものとしたら長さ $2L$ のとき 2^L 通りであり、実験でこの文字列を入力文字列として採用する。

実験では音訳の N ベスト候補を作成する時間、スペルチェックの候補作成時間（辞書に候補はあるかどうかを調べる時間を含めない）、入力から候補を表示する時間と入力文字列の長さの関係を計測してみる。結果を図 4.1、4.2、4.3 に示す。 図 4.1 から、音訳器により N ベスト候補

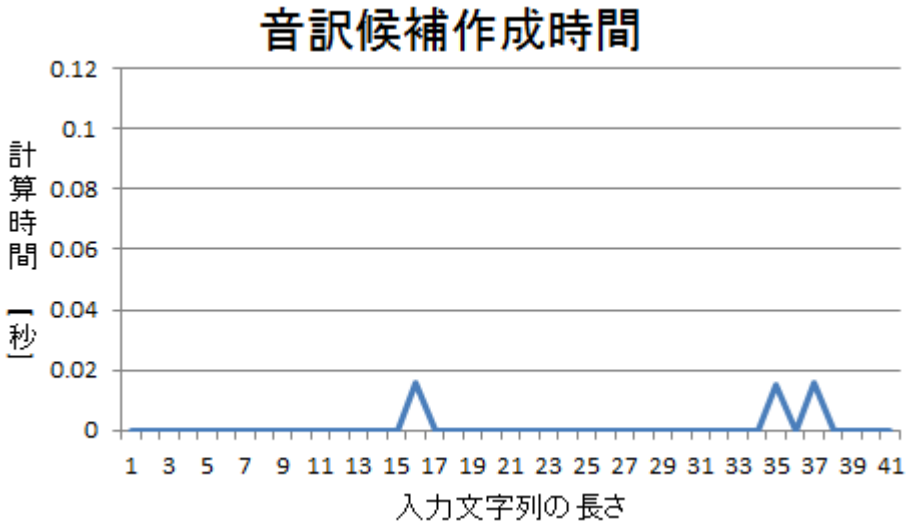

図 4.1: N ベスト候補を作成する時間

を作成する時間は入力文字列の長さに対して変化なく、ほぼ 0 秒であることがわかる。計算量は $O(N * 64L * log(64L))$ なので、定数項はほとんどないことがわかる。

図 4.2 と図 4.3 からから、スペルチェックの候補作成時間および入力から候補を表示するまでの時間は入力文字列の長さに比例して増えていることがわかる。また、入力から候補を表示するまでの時間の半分ほどがスペルチェックの候補作成にかかっていることがわかる。音訳候補作成時間はほぼ 0 秒だったので残りの半分は予測とスペルチェック候補を辞書にあるかどうかを調るのにかかったと思われる。予測は定数候補を調べているので、ほとんどの時間はスペルチェックにかかっていると言える。想定する誤りの回数を二つ以上にした場合かなり計算時間がかかることが予想される。しかし、全体として最大の長さでも計算時間は 0.12 秒なので提案手法では十分に高速であると言える。

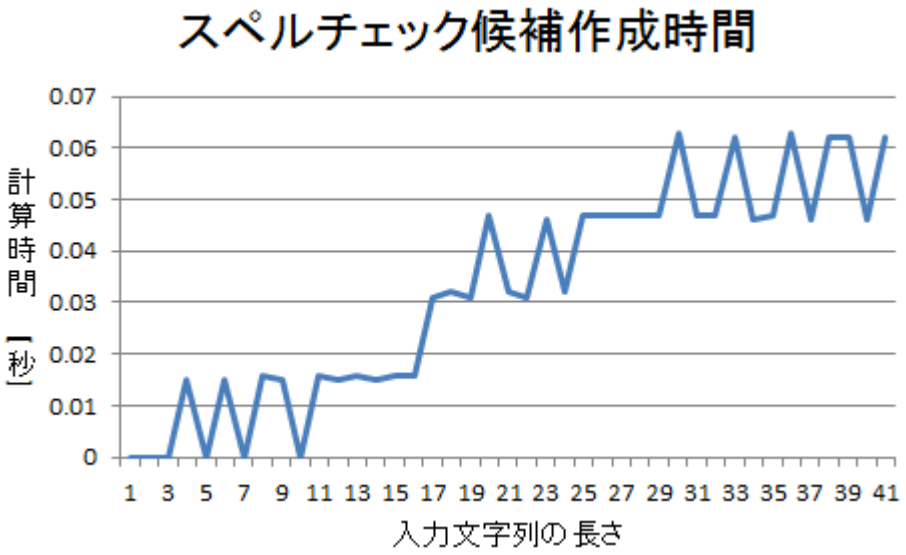

図 4.2: スペルチェックの候補作成する時間

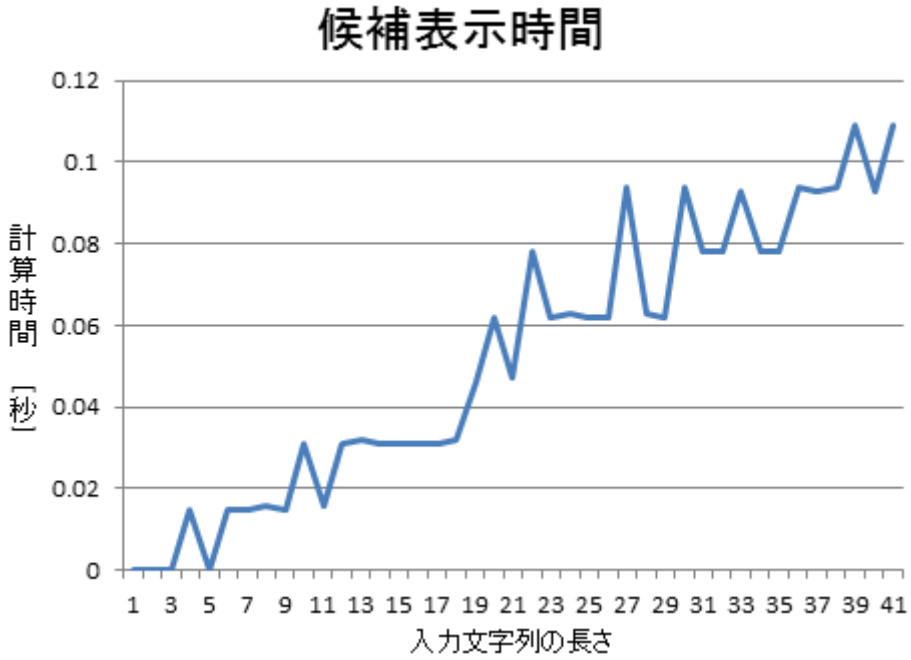

図 4.3: 入力から候補を表示するまでの時間

4.2 動作確認

モンゴル語 IME の動作確認環境（開発環境でもある）は以下の通りである。

- OS : Windows 7 (32bit)
- メモリ : 2.00GB
- CPU : Intel(R) Core(TM)2 Duo 2.00GHz

また、Windows 7 でモンゴル語 IME に切り替えられること、使用中にクラッシュはしなかったが、アプリケーションのメモリが増え続けていることがわかった。ここで、よく使う以下のアプリケーションでモンゴル語 IME を使用し、キリル文字入力可能か、候補表示するか、候補をマウスで選択できるか、候補をキーで選択できるかを確認した。

表 4.1: モンゴル語 IME の動作確認

アプリケーション	入力可能	候補表示	キー選択	マウス選択
WordPad	○	○	○	○
NotePad	○	○	○	○
Word	○	○	○	×
Excel	○	○	○	○
PowerPoint	○	○	○	○
Google chrome	○	○	○	×
Internet explorer	○	○	○	○
Yahoo messenger	○	○	○	○

結果としていくつかのアプリケーションでマウス選択できなかったが、基本機能は使えるため Windows 上のほとんどのアプリケーションと連動して動けると言える。

4.3 ユーザビリティテスト

実際に実装したモンゴル語 IME を留学生、社会人、定年者など含むモンゴル人 23 人に使ってもらい使用性を測ってみた。やり方は一定の文章を書いてもらいその後でモンゴル語 IME の持つ機能を説明し、もう一度好きな文章を書いてもらった。最後にアンケートを実施して、その平均をとった。アンケートの質問は以下の通りである。

- 普段パソコンで入力するときキリル文字の代わりにアルファベットで入力するのは何パーセントですか？
- 使ってみて 100 点満点で評価するとしたら何点ですか？
- 良かった点は何ですか？
- 悪かった、改良すべき点は何ですか？

- このソフトウェアを日常的に使いたくなりましたか？

アンケートの結果を図に示した。図 4.4 を見てみるとモンゴル語を入力するときはアルファベッ

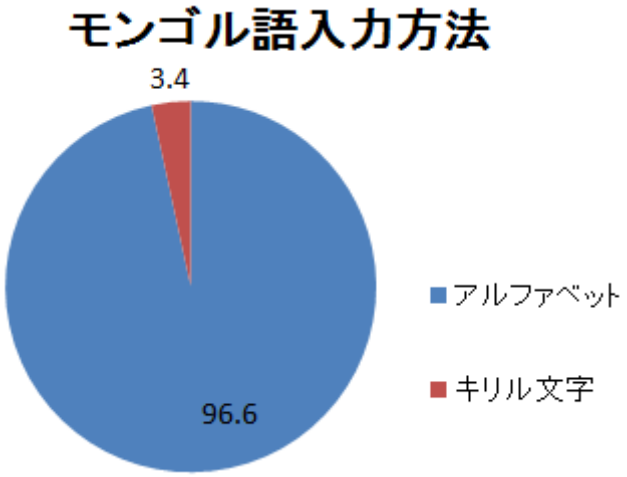

図 4.4: アルファベット入力とキリル入力の割合

トを用いて入力する方法全体の 96.6%になっている。この結果はキリル文字で入力するよりアルファベット入力してしまうというこの研究の背景と問題点を裏付けるものになっていることがわかる。

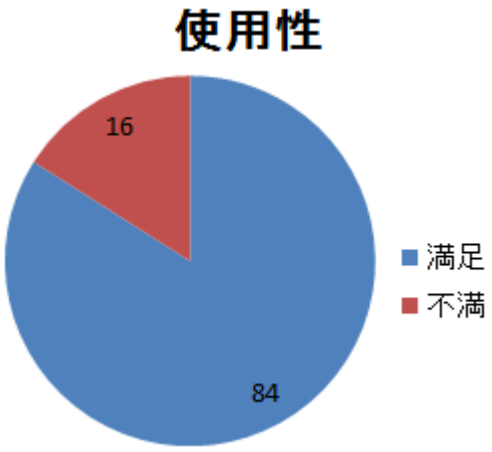

図 4.5: 使用性の評価

図 4.4 から分かることは実装したモンゴル語 IME を使ってみて良かった、満足したという人は 84%にもなっている。このソフトウェアを日常的に使いたくなりましたか？という質問には全員「はい」と答えた（図 4.6）。その中でも「ぜひぜひ」、「もちろん」という人もいた。

アンケートで良かった点と答えたものを以下にまとめた。

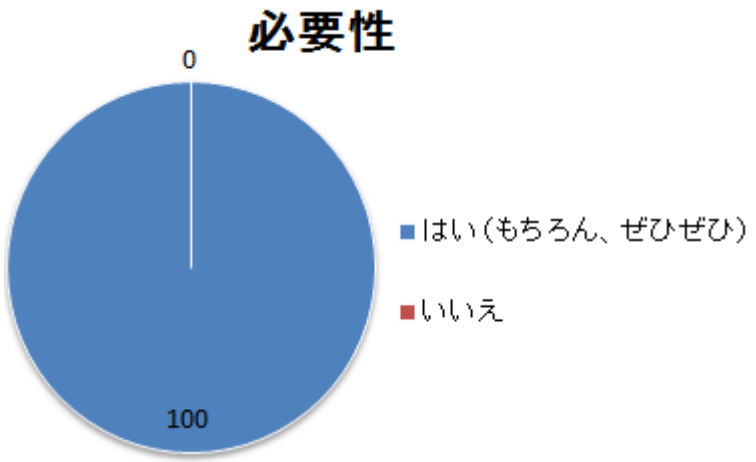

図 4.6: 必要性の評価

- モンゴル語で書ける (正しい単語を学習する）
- スペルチェック機能があって面白い、すごい
- 入力途中予測ができる
- 省略単語を登録できる
- こういうのは新しい

アンケートで悪かったあるいは改良すべき点と答えたものを以下にまとめた。

- 候補を数字で選択できたらいい
- 自動学習機能が欲しい
- Mac OS でも対応して欲しい
- Web 上のキリル音訳表を採用してほしい

日本語 IME などを使っているモンゴル人から見ると改良する点はいくつかあったが全体として実装した機能はすべてに気に入ってもらった。

以上の結果から実装した IME はモンゴル人に必要なもので使ってみて使用感も高かったといえる。

第5章

総括

5.1 まとめ

本研究では統計的なモデルを用いて、英語アルファベットからモンゴル語キリル文字に音訳し、更にスペルチェック、予測できる手法を提案し、最終形態として Windows 上で動作するモンゴル語入力ソフトを作った。モンゴル語入力の主な機能は以下の通りである。

- 追加される IME はモンゴル言語から選べられる
- 入力されるアルファベットキーをキリル字に音訳する。この時に他の手法と違って一つに絞らず N 個作っている。
- モンゴル語単語辞書ファイルを読み込み、単語を検索できる。
- 音訳器の結果に対してスペルチェックを行い、正しい単語を勧める。
- ユーザーの負荷を軽減するために入力の予測を行う
- 複数の候補を表示し、候補 Window から候補を選択できる
- Windows 上のの Application と連動して動く。

アンケートを実施することでアルファベットベースの入力方法の必要性を立証し、音訳でだれでもキリル文字入力できることを示し、問題点を解決できた。また、提案手法の検証により、モンゴル語 IME は十分な速さであり、Windows 上の様々なアプリケーションと連動して動くため実用性が高いといえる。

5.2 今後の課題

本研究で実装したモンゴル語 IME の実用性の高めるために考えられる必要な要素は以下の通りである。

- 辞書ファイルの充実化： 現在 6 万単語の辞書を用いているため単語の数を数百万まで増やす
- 辞書ファイルの圧縮、効率的な処理： データ構造のトライ木などを用いて辞書ファイルを圧縮する必要がある
- ユーザー入力履歴の活用：ユーザーの入力した単語などを覚えて次の時の予測、スペルチェックなどの使う
- IME の Bug 修正：モンゴル語 IME を色々なところで使ってみてプログラムのバグを修正する必要がある。

謝辞

本研究を進めるにあたり，多くの方からご指導，ありがとうございました．指導教官である笠井 裕之准教授には，研究内容から研究に対する姿勢まで，貴重なご指導をいただきました．私が充実した研究環境で，大学生活を送ることができたのは，笠井 裕之准教授のお陰であると実感しております．ここに深く感謝の意と礼儀を申し上げます．IME の実装方法、自然言語処理技術などについてアドバイスをした Tserenchimed Badarch さんに感謝いたします．研究室の優秀なメンバーらとは多くの時間を共有し，さまざまな刺激を受けました．研究をまとめあげられたのも，皆様の頑張る姿に背中押されていたからだと思います．研究室の同級生たちにも感謝します．最後まで諦めずに頑張る姿に刺激を受けました．また，私生活において心の支えとなってくれた友人たちにも深く感謝いたします．最後に，経済的にも精神的にも支えて下さった家族，応援や世話してくれた皆に心から感謝いたします．

参考文献

[1] 既存のモンゴル語入力方法

http://www.bolor-toli.com/

http://www.tsahimurtuu.mn/

http://dusal.blogmn.net/9815/

https://code.google.com/p/buuz/

http://www.medeelel.com/tools/cyrillic

[2] モンゴル語スペルチェック

http://spell.bolorsoft.com/

http://extensions.openoffice.org/en/project/mongolian-spell-checking-dictionary

http://dict.num.edu.mn/

[3] IME, 実装方法

http://ja.wikipedia.org/wiki/インプットメソッド

http://www.google.com/inputtools/windows/index.html

http://www.google.com/inputtools/help/languages.htmll

http://msdn.microsoft.com/en-us/library/windows/apps/ms629032.aspx

[4] 言語処理における統計的モデル

Tokunaga Hiroyuki, "日本語入力を支える技術　変わり続けるコンピュータと言葉の世界" WEB+DB PRESS plus

永田昌明: 統計的言語モデルと N-best 探索を用いた日本語形態素解析法, 情報処理学会論文誌, Vol. 40, No. 9, pp. 3420—3431 (1999).

Nagata, M. A Stochastic Japanese Mophological Analyzer Using a Forward-DP Backward-A* N Best Search Algorithm, Proceedings of the 15th International Conference on Computational Linguistics, pp. 201-207 (1994).

F. Song, W. Bruce Croft. CIKM '99 Proceedings of the eighth international conference on Information and knowledge management: A General Language Model for Information Retrieval

Charniak, E. Statistical Language Learning. The MIT Press, Cambridge MA, 1993

Manning, C., and Schtze, H. Foundations of Statistical Natural Language Processing. The

MIT Press, 1999

R. Baeza-Yates and B. Ribeiro-Neto, Modern Information Retrieval. Addison-Wesley, 1999.

[5] スペルチェック

How to Write a Spelling Corrector http://norvig.com/spell-correct.html

[6] Viterbi algorithm　ビタビアルゴリズム

http://ja.wikipedia.org/wiki/ビタビアルゴリズム

Andrew J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm, IEEE Transactions on Information Theory 13(2):260—269, April 1967.

G. D. Forney. The Viterbi algorithm. Proceedings of the IEEE 61(3):268—278, March 197

[7] A* algorithm （A-star, エースター）探索アルゴリズム

http://ja.wikipedia.org/wiki/A*

Hart, P. E.; Nilsson, N. J.; Raphael, B. (1968). ”A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. IEEE Transactions on Systems Science and Cybernetics SSC4 (2): pp. 100—107. PDF

[8] Spell check スペルチェック

Jurafsky, Daniel, and James H. Martin. 2009. Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics. 2nd edition. Prentice-Hall.

Chris Manning and Hinrich Schtze, Foundations of Statistical Natural Language Processing, MIT Press. Cambridge, MA: May 1999.

GNU Aspell http://aspell.net/

The LingPipe project’s spelling tutorial http://alias-i.com/lingpipe/demos/tutorial/querySpellChecker/read-me.html